Ntop, persistent data and rrd

What is ntop?

ntop is an open source network monitor. It collects information about the protocols used and hosts present on a network for display. ntop doesn't use snmp - it's not a device centric view of the network. Instead ntop actually processes the network packets directly.

While the processing of individual packets requires a lot more computer resources than just reading counters from devices such as routers, this gives much more detailed information - for example ntop sees the actual web server request instead of just that there was traffic on port 80. On the minus side, it's pretty easy to exceed the processing power of the low end machine typically available for ntop. An ISP using ntop to monitor a couple of T3s needs a FAST computer and A LOT of memory.

ntop also requires access to the physical network (either directly via a network card or indirectly via a netFlow/sFlow probe). This limits ntop's (usefullness|ability) to work across sites.

ntop doesn't care about the lowest (layer 1 or wire) layer. It leaves dealing with that to a library, libpcap, which hides most of that. ntop is designed as a hybrid packet analyzer, not a pure Ethernet analyzer (layer 2) nor a pure TCP/IP analyzer (layer 3).

ntop gets the data at the layer 2 (frame) level, which could be Ethernet or another protocol. Beyond Ethernet, ntop has minimal smarts about FDDI, PPP, RAW and TOKEN-RING frames. That is, at least enough for some basic counts or to extract the (layer 3) TCP/IP data in side.

ntop doesn't do a good job of showing multiple 'networks' - it's really focused on aggregating a picture of a single network. And for drilling down into that picture and presenting data recorded over long periods of time.

How can I store data between ntop runs?

Bluntly, you can't.

ntop uses it's rrd databases as the ONLY persistent storage. The data presented on the ntop web pages – other than the rrd based graphs – are generated from memory based tables and counters. When ntop restarts, that data is gone.

You can use the data extract facilities (Admin | Data Dump) to pull files of ntop's detailed counts on a periodic basis and pump those into say a mySQL database, but that's external to ntop. There are some scripts in the www directory, but they're unsupported.

Ntop, persistent data and rrd

What is RRD?

RRD stands for 'round-robin database". It is a special type of database designed for holding sequences of information over periods of time, without growing in size.

Rrdtool is a tool for manipulating RRDs. The home page for rrdtool is at http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/

What do they do? Well, suppose you want to compute the average of the traffic to your web site for the last fifteen minutes.

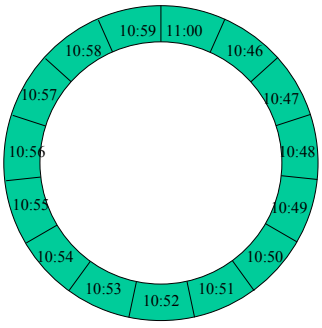
If you record the data each minute and save it in a traditional database it looks like this:

10:45	1.00 MB
10:46	1.02 MB
10:47	0.27 MB
…	
11:00	0.54 MB
…	

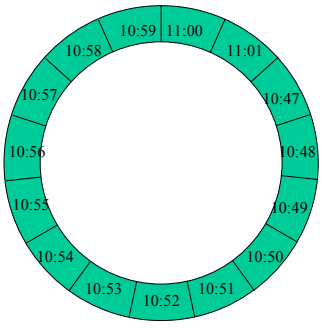
While you have the data to compute the total, the database grows in size forever. And all you really need are the last 15 values…

Certainly, you can create a purge routine and periodically remove the old data. But this type of constantly growing SQL database – even with a prune process – will require reorganization and rebuilds over time.

Suppose you had some kind of data structure where the last value was thrown away each time you added a new one – in Computer Science terms, a 'ring buffer'. Something that looks like this:



When it comes time to store the 11:01 value, you overlay the 10:46 value:



At any time, you still have the last 15 values. That – slightly simplified – is an RRD. The benefit is that your database never grows in size. The down side is that everything else in your history is gone – if your needs ever change, tough.

The ring buffers are called round-robin archives or RRAs. The RRA actually stores the RATE (bits per second), so the 10:47 value of 0.27 Megabytes is 0.27 * 1024*1024 * 8 / 60 or 37748.736 (bits per second). But it functions just like the rings described above.

ntop uses RRDs to store data over long periods of time. Separate files are created for each counter, in a structure that reflects the interfaces and hosts ntop sees. The specifics of what's recorded – interfaces or not, hosts or not, etc. is controlled by switches on the ntop rrd plugin.

If, for example, ntop's data (-p) is stored in /usr/share/ntop, then the rrd files will be in subdirectories under /usr/share/ntop/rrd/:

Graphics – used as a work area
Flows – used to store counters for packet flows into the plugins
Interfaces – subdirectories by interface name (e.g. eth0, sis0, Netflow-Device) hold the detailed counters for the interface itself and for the hosts seen on that interface.

For example:

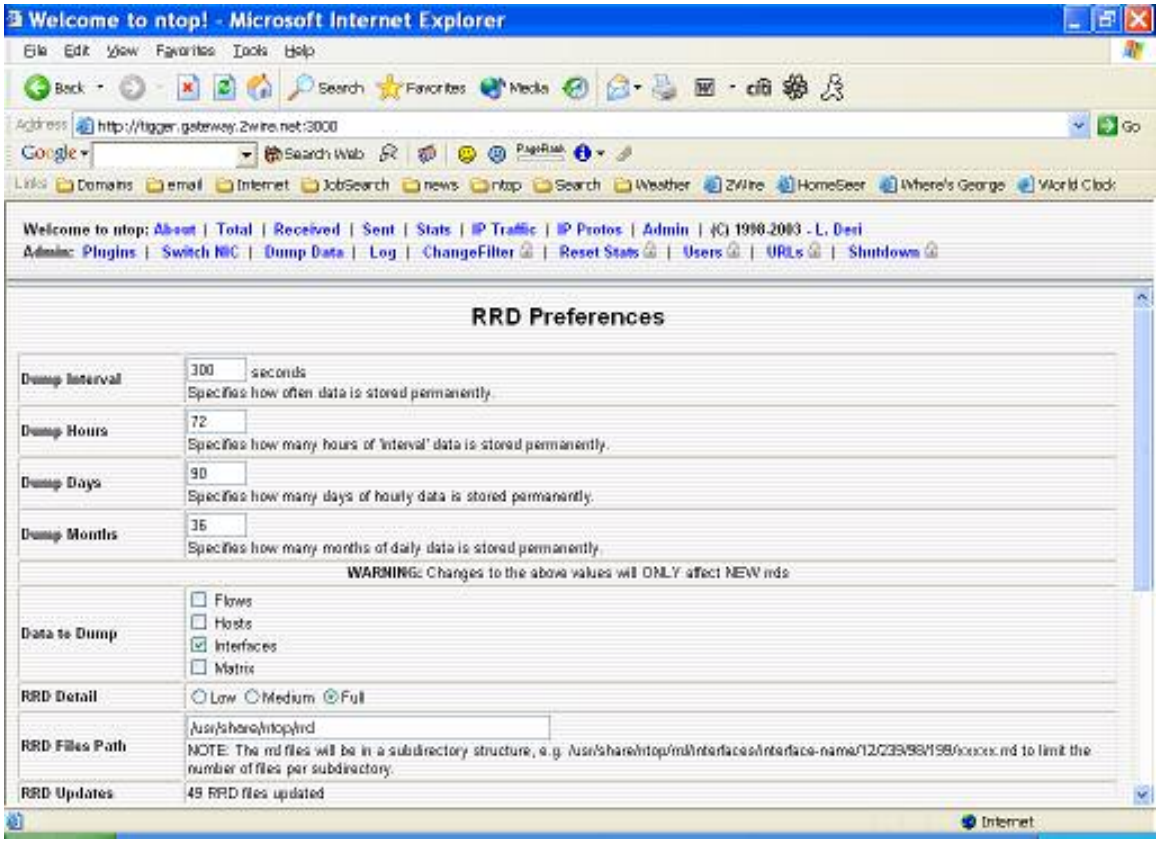
```
/usr/share/ntop/rrd//interfaces:
        eth0
                activeHostSendersNum.rrd
                arpRarpBytes.rrd
                broadcastPkts.rrd
                ethernetBytes.rrd
                ethernetPkts.rrd
                fragmentedIpBytes.rrd
                hosts
                        10
                                178
                                        0
                                                1
                                                        bytesSentLoc.rrd
                                                        bytesSent.rrd
                                                        icmpSent.rrd
                                                        …
                                248
                        12
                        …
                icmpBytes.rrd
                igmpBytes.rrd
                ipBytes.rrd
                IP_DHCPBytes.rrd
                IP_DNSBytes.rrd
                IP_FTPBytes.rrd
                …
                upTo512Pkts.rrd
                upTo64Pkts.rrd

        eth1

        …
```

There are many, many different counters that will appear if at least one packet is seen. For example, a host that never sends an ARP message will never have the arpRarpBytes.rrd counter.

To provide long term storage, ntop uses three interlocking rings. If you look at the ntop rrd plugin page, "RRD Preferences", you'll see the parameters for configuring the rings:



The configuration identifies how often and how much data is stored 'permanently':

Dump Interval	*Specifies how often data is stored.*
Dump Hours	*Specifies how many hours of 'interval' data are stored.*
Dump Days	*Specifies how many days of hourly data are stored.*
Dump Months	*Specifies how many months of daily data are stored.*

These four items define three interlocking rings. If you look ntop's log, you will see the rrd create line:

```
RRD: rrdtool create --start now-1 file --step 300
   DS:counter:COUNTER:300:0:12500000
   RRA:AVERAGE:0.5:1:864
   RRA:MIN:0.5:1:72
   RRA:MAX:0.5:1:72
   RRA:AVERAGE:0.5:12:2160
   RRA:AVERAGE:0.5:288:1080
```

This is all described on the man page for rrdcreate (man rrdcreate):

```
       DS:ds-name:DST:heartbeat:min:max
              A single RRD can accept input from several data sources (DS).
              (e.g. Incoming and Outgoing traffic on a specific communication
              line). With the DS configuration option you must define some
              basic properties of each data source you want to use to feed
              the RRD.

              ds-name is the name you will use to reference this particular
              data source from an RRD. A ds-name must be 1 to 19 characters
              long in the characters [a-zA-Z0-9_].

              DST defines the Data Source Type. See the section on "How to
              Measure" below for further insight.  The Datasource Type must
              be one of the following:
...
              COUNTER
                  is for continuous incrementing counters like the InOctets
                  counter in a router. The COUNTER data source assumes that
                  the counter never decreases, except when a counter over-
                  flows.  The update function takes the overflow into
                  account.  The counter is stored as a per-second rate. When
                  the counter overflows, RRDtool checks if the overflow hap-
                  pened at the 32bit or 64bit border and acts accordingly by
                  adding an appropriate value to the result.
...
              heartbeat defines the maximum number of seconds that may pass
              between two updates of this data source before the value of the
              data source is assumed to be *UNKNOWN*.

              min and max are optional entries defining the expected range of
              the data supplied by this data source. If min and/or max are
              defined, any value outside the defined range will be regarded
              as *UNKNOWN*. If you do not know or care about min and max, set
              them to U for unknown. Note that min and max always refer to
              the processed values of the DS. For a traffic-COUNTER type DS
              this would be the max and min data-rate expected from the
              device.

              If information on minimal/maximal expected values is available,
              always set the min and/or max properties. This will help RRD-
              tool in doing a simple sanity check on the data supplied when
              running update.
```

So "DS:counter:COUNTER:300:0:12500000" means we're defining a 'data source', named 'counter', which can go no more than 300 seconds between data points (otherwise they're 'unknown') and can have values from 0..12,500,000.

Our dump interval parameter indicates how frequent the slots in the ring are. The default is 300 seconds (5 minutes). This is both the –step value and the heartbeat in the DS definition.

Going back to the rrdcreate man page, we're defining three RRAs (ring buffers), according to this:

```
       RRA:CF:xff:steps:rows
              The purpose of an RRD is to store data in the round robin
              archives (RRA). An archive consists of a number of data values
              from all the defined data-sources (DS) and is defined with an
              RRA line.
```

```
When data is entered into an RRD, it is first fit into time
slots of the length defined with the -s option becoming a pri-
mary data point.

The data is also consolidated with the consolidation function
(CF) of the archive. The following consolidation functions are
defined: AVERAGE, MIN, MAX, LAST.

xff The xfiles factor defines what part of a consolidation
interval may be made up from *UNKNOWN* data while the consoli-
dated value is still regarded as known.

steps defines how many of these primary data points are used to
build a consolidated data point which then goes into the
archive.

rows defines how many generations of data values are kept in an
RRA.
```

The first RRA, `RRA:AVERAGE:0.5:1:864` means we're going to store 864 rows of data. Each row is a consolidation of 1 primary point (or a 5 minute interval). That 864 number comes from creating enough entries to store the 'Dump Hours' requirement. The default is 72 hours of 5 minute intervals, meaning 864 slots.

The ring concept means that the 865th value overlays the 1st. You always have the most recent 864, never more or less - although when you first create the RRA at time t, the 863 values for times less than t are 'unknown'.

Now you can DO updates (man rrdupdate) at less than 300-second intervals (all the way down to 1 second), but RRD will just combine them into that 300-second interval... (this time quoting man rrdupdate):

```
The update function feeds new data values into an RRD. The data gets
time aligned according to the properties of the RRD to which the data
is written.
```

I *think* this means that if you do this:

```
rrdtool update ipbytes.rrd 887457267:10
rrdtool update ipbytes.rrd 887457268:10
rrdtool update ipbytes.rrd 887457269:10
```

rrdcreate will actually update whatever row the 300 second interval for 88745726x falls into with (10+10+10)/ 300s or 0.1/second... but I'm not 100% sure. Anyway, ntop isn't SUPPOSED to make more than one update per interval. So it's not SUPPOSED to matter.

While that MIN and MAX look odd, I think it's because they're 2nd order values - i.e. the minimum of the 72 hourly entries (I know I tried to change it and it screwed everything up).

The second RRA stores hourly data (i.e. the sum of 12 5 minute values, as always in an RRA, normalized to a 1 second rate). Based on 'Dump Days' (default 30), there are 30 * 24 or 720 slots in the second RRA.

The third RRA stores daily data (i.e. the sum of 24 hourly values, normalized to the 1 second rate). Based on 'Dump Months' (default 36), there are 36 * 30 or 1080 slots in the third RRA.

Suppose the slots we record are:

13:05	300 packets
13:10	300 packets
13:15	600 packets
13:20	*Missing*
13:25	450 packets
13:30	*Missing*
13:35	*Missing*
13:40	*Missing*
13:45	*Missing*
13:50	*Missing*
13:55	*Missing*
14:00	600 packets

Our data 'rows' are now (remember rrd converts the absolute numbers into a per second value):

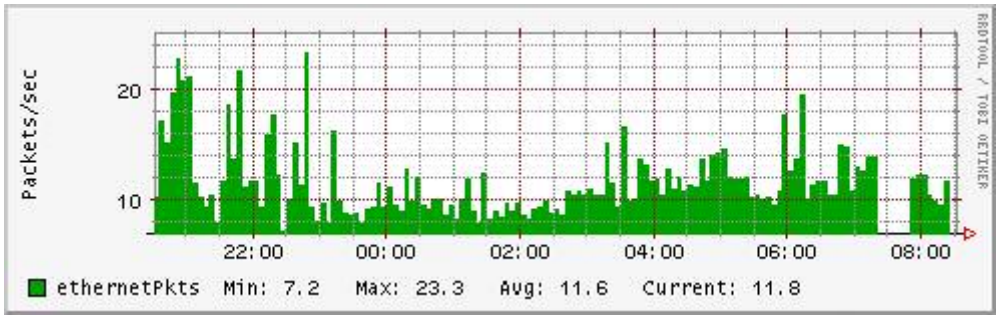
13:05	1.0
13:10	1.0
13:15	2.0
13:20	-
13:25	1.5
13:30	-
13:35	-
13:40	-
13:45	-
13:50	-
13:55	-
14:00	2.0

And so on, for the full 864 rows at 5 minute intervals (72 hours).

The second RRA, is a roll up of each 12 primary points (e.g. 60 minutes or 1 hour), and there are 2160 (90 days) worth. We average the primary points and no more than 50% (0.5) can be missing... That value can cause data to be 'lost' between RRAs.

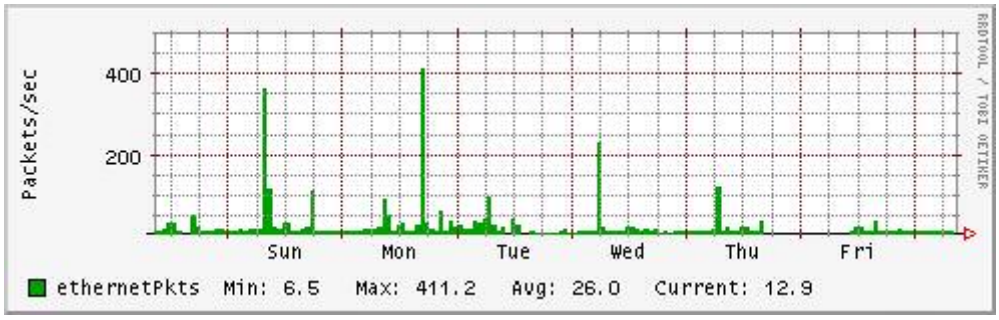
Continuing with our example, the 5 minute intervals shows 1.0 1.0 2.0 - 1.5 - - - - - - -, representing 2250 packets. However, 7 – which is more than 50% - of the 12 data points are missing. So the 1-hour interval shows *'missing'* - and those 2250 packets are 'lost'.

So what about the graphs? Let's look at the same data from different rings.

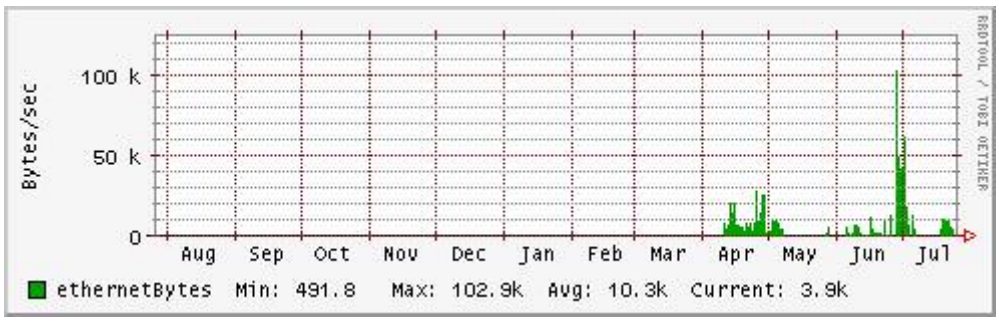


Although it's counter-intuitive (and AFAIK undocumented), rrdgraph does not use data from multiple RRAs in a single graph. From experimentation, it seems to pick the rrd that has the 'best' coverage. So say you're doing a 6-hour graph and have 3 hours of data in the 5 minute RRA and 4 hours of data in the 1 hour RRA. It will use the 4 hourly points.

So the single day graph, above is using the 5-minute interval data from the 1st RRA. The weekly graph below – of the 'same' item – is using the hourly interval data from the 2nd RRA.



And this third graph is using the daily data:



In the legends, the 'k', 'm' or 'g', as in 3.9k means what you think – 'kilo', 'mega' or 'giga' as in 3.9k = 3.9 * 1024 or 3993 bytes / second.

What data is Dumped into RRDs? Is there a list?

Nope. The authoritative list is the source – see the code. The routine is rrdMainLoop() in plugins/rrdPlugin.c, where you'll see the lists for the various types of dumps – if(dumpHosts), if(dumpFlows), if(dumpInterfaces)and if(dumpMatrix), where you'll see the individual counters listed, such as

```
updateTrafficCounter(rrdPath, "pktDuplicatedAckSent", &el->pktDuplicatedAckSent);
updateTrafficCounter(rrdPath, "pktDuplicatedAckRcvd", &el->pktDuplicatedAckRcvd);
```

It's really not hard to figure them out – parse them into words at the capitals and apply a little knowledge of the various protocols (google for them by name if nothing else or use a reference book). So, for example, arpReplyPktsRcvd.rrd is

arp (which is a protocol) Reply Pkts (Packets) Rcvd (Received)

arpReplyPktsSent is

arp reply packets sent

etc.

If there's a specific one you can't figure out, grep in the source for the name, check the counter and see what it's counting...

```
$ grep --line-number arpReplyPktsSent plugins/*.[ch]

returns:

plugins/rrdPlugin.c:            updateTrafficCounter(rrdPath, "arpReplyPktsSent",
&el->arpReplyPktsSent);
```

So

```
$ grep --line-number arpReplyPktsSent *.[ch]

returns

emitter.c:829:      if(checkFilter(filter, "arpReplyPktsSent"))
emitter.c:830:  wrtLlongItm(fDescr, lang, "\t", "arpReplyPktsSent", el-
>arpReplyPktsSent, ',', numEntries);
globals-structtypes.h:578:  TrafficCounter   arpReqPktsSent, arpReplyPktsSent,
arpReplyPktsRcvd;
pbuf.c:2537:          if(srcHost != NULL) incrementTrafficCounter(&srcHost-
>arpReplyPktsSent, 1);
reportUtils.c:2006: if(el->arpReqPktsSent.value+el->arpReplyPktsSent.value+el-
>arpReplyPktsRcvd. value > 0) {
reportUtils.c:2038:             formatPkts(el->arpReplyPktsSent.value)) < 0)
```

The key routines are usually pbuf (processes the packet buffer) and sessions (processes tcp/ip sessions). So, open up pbuf.c and look at line 2537...

The code really isn't THAT hard to read for this stuff - just ignore the C-ish stuff and read it in pseudo-English, checking against a protocol reference book if necessary.

So:

```
        switch(eth_type) {
        case ETHERTYPE_ARP: /* ARP - Address resolution Protocol */
          memcpy(&arpHdr, p+hlen, sizeof(arpHdr));
          if(EXTRACT_16BITS(&arpHdr.arp_pro) == ETHERTYPE_IP) {
            int arpOp = EXTRACT_16BITS(&arpHdr.arp_op);

            switch(arpOp) {
            case ARPOP_REPLY: /* ARP REPLY */
              memcpy(&addr.s_addr, arpHdr.arp_tpa, sizeof(addr.s_addr));
              addr.s_addr = ntohl(addr.s_addr);
              dstHost = lookupHost(&addr, (u_char*)&arpHdr.arp_tha, 0, 0,
actualDeviceId);
              memcpy(&addr.s_addr, arpHdr.arp_spa, sizeof(addr.s_addr));
              addr.s_addr = ntohl(addr.s_addr);
              srcHost = lookupHost(&addr, (u_char*)&arpHdr.arp_sha, 0, 0,
actualDeviceId);
              if(srcHost != NULL)
incrementTrafficCounter(&srcHost->arpReplyPktsSent, 1);
              if(dstHost != NULL)
incrementTrafficCounter(&dstHost->arpReplyPktsRcvd, 1);
              /* DO NOT ADD A break ABOVE ! */
            case ARPOP_REQUEST: /* ARP request */
```

```
                memcpy(&addr.s_addr, arpHdr.arp_spa, sizeof(addr.s_addr));
                addr.s_addr = ntohl(addr.s_addr);
                srcHost = lookupHost(&addr, (u_char*)&arpHdr.arp_sha, 0, 0,
actualDeviceId);
                if((arpOp == ARPOP_REQUEST) && (srcHost != NULL))
incrementTrafficCounter(&srcHost->arpReqPktsSent, 1);
              }
```

Becomes

switch on EtherType (which is a field defined in the Ethernet protocol)

```
  if it's ARP REPLY
       (Have a source host)? increment traffic counter for arpReplyPktsSent
       (Have a destination host)? increment traffic counter for
arpReplyPktsRcvd
  if it's ARP REQUEST (also fall in here from above because there's no
break)
       (ARP REQUEST? and Have a source host?)? increment traffic counter for
arpReqPktsSent
```

All the C you need to know is that switch(something) case value is like asking "Does 'something' have the value 'value'". And that 'break' means I'm done with that question, let's ask another...

Try it! You'll be surprised how easy it is to follow, especially in pbuf.c - which is long, but it's really just endless lists of "if something is this, do this and count that, otherwise if it's that, do this other thing, otherwise ..."

Dumping Data from an RRD

There is a dump option rrdtool, which dumps the contents of an RRD as an file in XML format:

```
$ rrdtool dump ……rrd
```

Which gives an xml-like file as output. Be aware of some issues:

- The generated file does not conform to xml standards – it's missing headers.
- No DTD is provided, although a seemingly workable one is available here: http://www.ee.ethz.ch/~slist/rrd-users/msg05412.html.
- The row time stamps are comments instead of data fields.

Past those issues, any tool capable of manipulating xml should find the dump files palatable. Here's a cut down example:

```
<!-- Round Robin Database Dump -->
<rrd>
        <version> 0001 </version>
        <step> 300 </step>
        <!-- Seconds -->
        <lastupdate> 1056806565 </lastupdate>
        <!-- 2003-06-28 08:22:45 CDT -->
        <ds>
```

```
                <name> counter </name>
                <type> GAUGE </type>
                <minimal_heartbeat> 300 </minimal_heartbeat>
                <min> 0.0000000000e+00 </min>
                <max> NaN </max>
                <!-- PDP Status -->
                <last_ds> UNKN </last_ds>
                <value> 5.2800000000e+03 </value>
                <unknown_sec> 0 </unknown_sec>
        </ds>
        <!-- Round Robin Archives -->
        <rra>
                <cf> AVERAGE </cf>
                <pdp_per_row> 1 </pdp_per_row>
                <!-- 300 seconds -->
                <xff> 5.0000000000e-01 </xff>
                <cdp_prep>
                        <ds>
                                <value> NaN </value>
                                <unknown_datapoints> 0 </unknown_datapoints>
                        </ds>
                </cdp_prep>
                <database>
                        <!-- 2003-06-25 08:25:00 CDT / 1056547500 -->
                        <row>
                                <v> NaN </v>
                        </row>
…
                        <!-- 2003-06-26 08:10:00 CDT / 1056633000 -->
                        <row>
                                <v> 7.9000000000e+00 </v>
                        </row>
                </database>
        </rra>
        <rra>
                <cf> MIN </cf>
                <pdp_per_row> 1 </pdp_per_row>
                <!-- 300 seconds -->
                <xff> 5.0000000000e-01 </xff>
                <cdp_prep>
                        <ds>
                                <value> NaN </value>
                                <unknown_datapoints> 0 </unknown_datapoints>
                        </ds>
                </cdp_prep>
                <database>
                        <!-- 2003-06-28 02:25:00 CDT / 1056785100 -->
                        <row>
                                <v> 1.0650000000e+01 </v>
                        </row>
…
                </database>
        </rra>
        <rra>
                <cf> MAX </cf>
                <pdp_per_row> 1 </pdp_per_row>
                <!-- 300 seconds -->
                <xff> 5.0000000000e-01 </xff>
                <cdp_prep>
                        <ds>
                                <value> NaN </value>
                                <unknown_datapoints> 0 </unknown_datapoints>
                        </ds>
                </cdp_prep>
                <database>
                        <!-- 2003-06-28 02:25:00 CDT / 1056785100 -->
                        <row>
                                <v> 1.0650000000e+01 </v>
                        </row>
…
```

```
                </database>
        </rra>
        <rra>
…
        </rra>
        <rra>
…
        </rra>
</rrd>
```

Problems

ntop eats a lot of space in the /usr/share/ntop/rrd directory

Yeah, it does.

With the default settings, about 35,292 bytes per counter. And there are lots of counters per host or interface. The usage patterns of the network strongly influence the space needed for the rrds. If you're running things that contact or are contacted by lots of hosts (e.g. a web server, P2P download site, use nmap, etc.), you will create records for a lot of hosts.

For example, these measurements come from a small home LAN, on which I don't run port scans nor use P2P software:

```
# cd /usr/share/ntop/rrd/interfaces/
# du -h --max-depth=1
3.7G    ./eth1
2.3G    ./eth2
729M    ./NetFlow-device
1017M   ./eth0
7.7G    .
```

If ntop is monitoring an even moderately busy system there can be a huge number of hosts – many 1000s or even millions.

That's why we use the a/b/c/d subdirectory structure to limit the number of files per level in the directory structure. Before that change was made, a number of systems easily exceeded the Linux limit of 32K files in a subdirectory.

Either turn off the plugin if you don't want it, or use the configuration parameters on the plugin to:

- Reduce the amount of data stored for future rrds.
- Set white/black lists to control which hosts are stored in rrds.

Can I purge files?

Sure – just remove the files or entire directory. When displaying the host details, ntop looks in the appropriate subdirectory. If it exists and has files, then the icon for the rrd graphs is displayed. If it's not present, the icon just doesn't appear.

I have old layout files, can I move them around?

Sure – they are just files, so all the normal tools (cp, mv, rm) work. If you move them into the 'right' place, ntop will find them and continue updating them. If ntop doesn't find them, it will create a new file – and there aren't any tools I'm aware of to combine files.

There's also a (*nix) script posted in the user-contributed section on SourceForge, called adjustrrdmodel. This shell script moves the rrd database files from their old positions (e.g. 192.168.1.1/xxxx.rrd) into new ones (e.g. 192/168/1/1/xxxx.rrd). Run with ntop down, in the rrd/<interface>/hosts directory.

> **WARNING: Have a backup before you run this and read the output messages.**
> **STATUS: Unsupported, provided AS IS! It's only been lightly tested.**
> **REQUIRES: bash, gawk**